# Big data clustering based on spark chaotic improved particle swarm optimization

**Saida Ishak Boushaki[1], Brahim Hadj Mahammed[1], Omar Bendjeghaba[2], Messaoud Mosbah[1]**
[1]Department of Computer Science, University M'Hamed Bougara Boumerdes, Boumerdes, Algeria
[2]Department of Automation and Electrification, Faculty of Hydrocarbons and Chemistry, University M'Hamed Bougara Boumerdes, Boumerdes, Algeria

**Article Info**

***Article history:***

Received Nov 4, 2023
Revised Dec 30, 2023
Accepted Jan 3, 2024

***Keywords:***

Big data clustering
Chaotic map
MapReduce
Particle swarm optimization
Spark

**ABSTRACT**

In recent years, the surge in continuously accelerating data generation has given rise to the prominence of big data technology. The MapReduce architecture, situated at the core of this technology, provides a robust parallel environment. Spark, a leading framework in the big data landscape, extends the capabilities of the traditional MapReduce model. Coping with big data, especially in the realm of clustering, requires more efficient techniques. Meta-heuristic-based clustering, known for offering global solutions within reasonable time frames, emerges as a promising approach. This paper introduces a parallel-distributed clustering algorithm for big data within the Spark Framework, named Spark, chaotic improved PSO (S-CIPSO). Centered on particle swarm optimization (PSO), the proposed algorithm is enhanced with a chaotic map and an efficient procedure. Test results, conducted on both real and artificial datasets, establish the superior performance and quality of clustering results achieved by the proposed approach. Additionally, the scalability and robustness of S-CIPSO are validated, demonstrating its effectiveness in handling large-scale datasets.

***Corresponding Author:***

Saida Ishak Boushaki
Department of Computer Science, University M'Hamed Bougara Boumerdes
Boumerdes, Algeria
Email: s.boushaki@univ-boumerdes.dz

## 1. INTRODUCTION

Recently, more than two exabytes of data are generated world wide every day, leading to the emergence of big data [1]–[7]. Consequently, the analysis of this data necessitates the development of more efficient and appropriate software [3]–[7]. Clustering, a crucial machine learning method, is widely utilized to extract meaningful information from vast volumes of data across various research areas. Its primary task is to unsupervisedly group the data collection so that elements within the same group are highly similar and homogeneous [8]–[12].

The principal challenge faced by big data clustering approaches is to scale up and speed up clustering algorithms without compromising clustering quality. These approaches are categorized into two classes: single-machine and multi-machine clustering. In the first class, sampling techniques are employed to reduce the size of datasets. Additionally, projection, sub-space clustering, and co-clustering methods are utilized to reduce the dimensionality of the datasets. While approaches in this class are valuable, they have their own set of shortcomings. For instance, they are constrained by limited memory, their processing time is time-consuming, they are sensitive to noise and outliers, and dimensionality reduction can impact the effectiveness of clustering algorithms [8], [9], [13]–[15].

Multi-machine clustering approaches follow a common process: initially, the data is partitioned into sub-groups and distributed across machines. Subsequently, each machine independently performs clustering on its allocated data partition. These approaches can be broadly categorized into two main types: parallel and MapReduce. Parallel algorithms, though potent for specific applications, may demand additional memory to store intermediate results or facilitate communication between processes. In contrast, MapReduce is particularly well-suited for processing extensive datasets distributed across a cluster. It gained popularity during the big data era due to its scalability and fault-tolerant design, offering a high-level abstraction for distributed computing [8], [9], [13]–[15].

The storage and processing of multi-machine clustering approaches have been executed using various frameworks, such as Hadoop, Spark, Storm, Flink, and Samza [16]–[21]. Recent research indicates the notable performance of the Spark Framework in enhancing the MapReduce model to support both interactive and iterative queries. Moreover, to accelerate calculations, Spark incorporates robustly distributed datasets for in-memory processing [22]–[24].

Multi-machine clustering algorithms constitute a thriving area of study, evolving conventional clustering methods to address the challenges posed by big data. A notable subset of these algorithms focuses on the well-known K-means approach. For example, Cui *et al.* [25] examined the challenges associated with processing big data using K-means in the context of MapReduce. Alguliyev and others have presented a new scalable K-means algorithm by splitting a dataset into batches [26]. Despite experimental tests on diverse datasets demonstrating the practical applicability of big data clustering based on K-means, these approaches contend with the primary drawback inherent to K-means being prone to getting trapped in local solutions.

In contrast, metaheuristic algorithms have demonstrated their efficiency in attaining global solutions within a reasonable time frame for various optimization problems [27]–[29], including big data clustering. For example, Benmounah *et al.* [30] proposed a scalable differential evolution (DE) algorithm based on the MapReduce programming model. Hashemi *et al.* [31] described a new algorithm for clustering big data, focusing on PSO. Nevertheless, a primary challenge associated with this class of algorithms is the issue of slow convergence.

To address this challenge, algorithms within this class are often hybridized with the well-known K-means algorithm. Sherar and Zulkernine [32] proposed a hybrid approach to big data clustering, combining K-means and particle swarm optimization (KMPSO), and implemented it in the Apache Spark Framework. Their work demonstrated the suitability of Apache Spark for handling large-scale datasets. Sinha and Jana [33] presented a novel hybrid approach incorporating genetic algorithms (GA) and K-means++. Alguliyev *et al.* [34] proposed a novel method for anomaly detection in big data, focusing on PSO and K-means algorithms. In the same year, Moslah *et al.* [35] introduced a big data clustering algorithm based on an adapted version of PSO combined with K-means, implemented under the Spark Framework (S-PSO). Ravuri and Vasundra [36] presented a novel MapReduce approach based on moth-flame optimization, the Bat algorithm, and sparse fuzzy C-means. Additionally, Bashabsheh *et al.* [37] presented a hybrid method focusing on harris hawk's optimizer (HHHO) and K-means clustering under the MapReduce framework. Although these algorithms leverage the advantages of the combined approaches, they often require significant computational resources and calculus due to their complexity.

Rather than advocating the combination of different algorithms, alternative methods leverage novel ideas derived from contemporary theories, such as chaos theory, to enhance their efficacy in a straightforward manner [38]–[40]. A promising approach to model systems heavily dependent on initial conditions, such as meta-heuristics, is through the use of chaotic maps. Chaotic variables, known for their excellent performance attributes, including ergodicity and non-repetition, replace random ones in this context [41].

The current study contributes to enhance the performance capabilities of existing big data clustering algorithms. The proposed approach is grounded in the well-known metaheuristic PSO [42] and is executed under the Spark Framework [43], [44]. In this context, the conventional PSO is improved by incorporating the properties of chaotic maps, specifically ergodicity and non-repetition. Additionally, an efficient procedure is implemented to accelerate the convergence speed of the search process. The results of the proposed approach are found to be satisfactory, both in terms of scalability and robustness, as well as in the performance and quality of clustering results. The structure of this paper is organized as: section 2 outlines the details of the proposed approach and the theories related to the algorithm, section 3 provides details of experimentation and results, and finally, conclusions and future work are summarized in the last section 4.

## 2. METHOD

In this study, the spark framework is chosen for its numerous advantages [20]–[24]. The proposed approach is grounded in the renowned PSO meta-heuristic [42], [45], [46]. The research demonstrates that this evolutionary technique is particularly well-suited for addressing distributed clustering problems. In this work, the search procedure is improved by leveraging the ergodicity and non-repetition performance properties of chaotic maps. Enhanced speed is achieved through the incorporation of the gravity center step. The subsequent section will delve into the concepts associated with chaotic maps.

### 2.1. Chaotic maps

A mathematical function that exhibits chaotic behavior over time is referred to as a chaotic map [41]. A chaotic sequence is generated by the values produced by various chaotic functions. In a chaotic sequence, the value at instant k+1 depends solely on the value at instant k. Various chaotic maps exist, including the circular map, gaussian map, logistic map, sinusoidal map, and tent map. Table 1 provides definitions for some well-known chaotic maps.

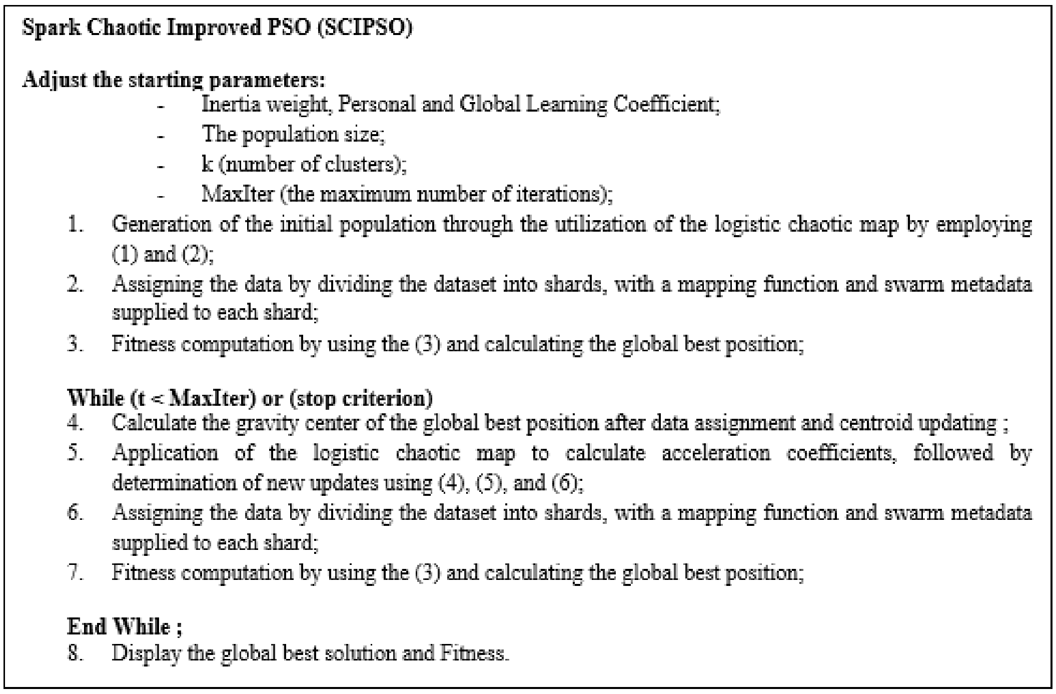
It's noteworthy that many optimization algorithms involve the manipulation of at least one random variable. However, chaotic optimization algorithms replace these random variables with chaotic variables to leverage the robust properties of chaotic maps, such as non-repetition and ergodicity. This substitution allows for the execution of global searches at rates faster than stochastic searches [41].

Table 1. Definition of chaotic maps

| Chaotic map | Definition |
|---|---|
| Circle map | $c_{k+1} = c_k + b - \left(\frac{a}{2\pi}\right)\sin(2\pi k)\, mod(1), a = 0.5\ et\ b = 0.2$ |
| Logistic map | $c_{k+1} = ac_k(1 - c_k), a = 4$ |
| Gauss map | $c_{k+1} = \begin{cases} 0 & if \quad c_k = 0 \\ 1/c_k\, mod(1) & \end{cases}$ |
| Tent map | $c_{k+1} = \begin{cases} c_k/0.7 & c_k < 0.7 \\ 10/3 & c_k \geq 0.7 \end{cases}$ |
| Sinusoidal map | $c_{k+1} = ac_k^2 \sin(\pi c_k), a = 2.3$ |

### 2.2. The steps of the proposed approach

The proposed algorithm, named Spark, chaotic improved PSO (S-CIPSO), involves four main steps: the generation of the initial population using chaotic maps, the assignment of data points to the nearest cluster, and the calculation of both fitness and the global best position. Additionally, it includes the calculation of the gravity center of the best global best position. Finally, the centroids and particle speed are updated using chaotic map. The pseudocode for S-CIPSO is presented in Figure 1.



```
Spark Chaotic Improved PSO (SCIPSO)

Adjust the starting parameters:
        -   Inertia weight, Personal and Global Learning Coefficient;
        -   The population size;
        -   k (number of clusters);
        -   MaxIter (the maximum number of iterations);
    1.  Generation of the initial population through the utilization of the logistic chaotic map by employing
        (1) and (2);
    2.  Assigning the data by dividing the dataset into shards, with a mapping function and swarm metadata
        supplied to each shard;
    3.  Fitness computation by using the (3) and calculating the global best position;

    While (t < MaxIter) or (stop criterion)
    4.  Calculate the gravity center of the global best position after data assignment and centroid updating ;
    5.  Application of the logistic chaotic map to calculate acceleration coefficients, followed by
        determination of new updates using (4), (5), and (6);
    6.  Assigning the data by dividing the dataset into shards, with a mapping function and swarm metadata
        supplied to each shard;
    7.  Fitness computation by using the (3) and calculating the global best position;

    End While ;
    8.  Display the global best solution and Fitness.
```

Figure 1. Pseudo code of the proposed S-CIPSO

### 2.2.1. Phase 1: Generation of the initial population

Most conventional metaheuristic algorithms typically generate their initial population randomly from the search space. However, in this study, the information for the initial population is created by employing the chaotic map. It should be noted that the logistic chaotic map is chosen based on empirical study. The positions of the particles represent the centroids of the initial clusters. The following formulas are used to generate the initial centroids:

$$\mathrm{z_{ij}} = \mathrm{Ub} + (\mathrm{Ub} - \mathrm{Lb}) * \mathrm{Cr(t)} \quad (1)$$

$$Cr(t+1) = 4 * Cr(t) * (1 - Cr(t)) \quad (2)$$

where:
$\mathrm{z_{ij}}$: is the j th centroid of the i th particle.
Lb: vector of lower limits, Ub: vector of upper limits, Cr: represents a logistic chaotic map.

### 2.2.2. Phase 2: Data assignment, fitness computation, and calculating the global best position

This phase is the most resource-intensive, as each particle requires a substantial amount of data to be assigned to nearby clusters. However, object assignment can be performed concurrently since it is independent of other objects. The dataset is initially divided into shards, with each shard receiving a mapping function and swarm metadata. Subsequently, the nearest cluster is determined for each data point using the mapping function. The mapping function then generates a key-value pair, where the key consists of the particleID and centroidID pair, and the value represents the shortest distance between a data object and its centroidID for the given particleID. It is important to note that the reduction function, referred to as the fitness computation step, utilizes the reduceByKey() operation provided by the Spark Framework. This operation combines the outputs of various mapping functions once all data has been assigned to the closest cluster. The following formula, aimed at minimizing the average distance between data objects and their centroids, is employed to determine the new fitness value:

$$Fit_i = \frac{\sum_{i=1}^{k}\left(\sum_{j=1}^{n} m_{ij} d(C_i, O_j)/n_i\right)}{k} \quad (3)$$

where:
$Fit_i$ denotes the fitness value of particle i, k indicates the number of clusters, n is the number of data objects, $m_{ij}$ =1 if the jth object is in the ith cluster and 0 otherwise. $d(C_i, O_j)$ is the Euclidean distance between the data object $O_j$ and the centroid of the cluster $C_i$, $n_i$ is the number of objects that belong to the cluster number i. Subsequently, the reduce function produces a key-value pair, where the updated fitness value serves as the value, and the particleID serves as the key.

– Let O = $\{O_1 ... O_n\}$ be the data objects and PI(t) = $\{PI_1(t), PI_2(t) ... PI_l(t)\}$ the particle information set at time t and l is the population size.
– $PI_i(t) = \{z_i(t), v_i(t), pbest\mathrm{PI}_i(t), pbestF_i(t)\}$ represents the information of particle i in iteration t where $z_i(t)$ is its position, $v_i(t)$ is its speed, $pbestP_i(t)$ is its best position, and $pbestF_i(t)$ is its best Fitness.
– Let $F = \{F_1 ... F_l\}$ be the set of fitness values where $F_i$ is the Fitness value of particle $i$.

It should be noted that the calculated fitness of the new particles is automatically stored in a resilient distributed dataset (RDD), from which it is later collected. Afterwards, each particle maintains its individual best position. The particle with the best fitness value (the minimum average distance between data objects and their centroids) is determined to be the gbest particle (gbest) for the update.

### 2.2.3. Phase 3: Calculation of the gravity center of the global best position

This stage involves two primary steps: data assignment and centroid updating. To allocate data objects to the nearest cluster, the data assignment phase employs a mapping function that takes the data fragment and centroids clusters from the preceding step as inputs. As a result, a list of key/value pairs is generated, where the key represents the cluster index to which the data item is allocated, and the value represents the output data vector. During the centroid calculation phase, the centroids of the clusters are updated by computing the average value for each cluster and aggregating the outputs of various mapping functions.

### 2.2.4. Phase 4: Update of centroids and particle speed using chaotic map

This step commences with the distribution of particle information among various mapping functions to facilitate the update of particle velocity and position, employing (4)-(6). It’s noteworthy that a logistic chaotic map is employed to update the acceleration coefficients, c1 and c2, of the particles. This incorporation ensures dynamic adjustments to the acceleration coefficients based on the characteristics of the chaotic map.

$$V_i(t+1) = w * V_i(t) + c1 * Cr(t) * (pbestP_i(t) - z_i(t)) + c2 * (1 - Cr) * (pbestP_i(t) - z_i(t)) \quad (4)$$

$$z_i(t+1) = z_i(t+1) + V_i(t) \quad (5)$$

$$Cr(t+1) = 4 * Cr(t) * (1 - Cr(t)) \quad (6)$$

Where,
$V_i(t)$ is the speed of particle i at time t, w denotes the inertia coefficient, $z_i(t)$ is the position of particle i at time t, c1 and c2 are the two acceleration coefficients.
$Cr(t)$ is represented the chaotic value at time $t$, $Cr(t+1)$ is represented the chaotic value at time $(t+1)$.
The reduce function employs the reduceByKey() method to consolidate the output from each mapping function into a unified RDD.

## 3. RESULTS AND DISCUSSION

### 3.1. Development environment

In this section, we introduce the development environment employed for implementing this research. Firstly, the hardware configuration was executed on a virtual machine, and its specifications are outlined in Table 2. Subsequently, the software configuration was undertaken utilizing the Apache Spark framework (version 3.0.0), and the implementation was executed in the Python programming language (version 3.7) within a Linux Ubuntu environment. To implement on Linux, the Spark Framework was installed and configured through the following steps:
- Installation of the Java program.
- Downloading the chosen Spark version from the official Spark website and subsequent installation.
- Configuration of environment variables, including setting the JAVA_HOME variable to specify the Java installation directory, specifying the SPARK_HOME variable, and configuring the PATH environment variable to allow the operating system to locate the Spark executable.

Table 2. Hardware specification

| Device | Specification |
|---|---|
| Hard disk | 100 GB |
| RAM | 16 GB |
| System type | 64-bit operating system |
| Processor | Intel(R) Core (TM) i5-6200U CPU. (5 cores) |
| Frequency | 2.30 GHz |
| Cloud service | Azure synapse analytics |

### 3.2. Datasets

The tests were conducted on both artificial and real datasets. Five series of large-scale artificial datasets were generated with a mean of 350 and a sigma of 100 using the Gaussian distribution. These datasets vary in size, ranging from 1 million to 5 million data points, with each data point represented by ten variables. The simulated data collections are denoted as D1M, D2M, D3M, D4M, and D5M, corresponding to 1,000,000; 2,000,000; 3,000,000; 4,000,000; and 5,000,000 data points, respectively. Additionally, three real datasets obtained from the UCI machine learning repository were used:
- Magic: Captures outcomes from simulating high-energy gamma particles using an atmospheric Cherenkov telescope on Earth. The dataset consists of 19,020 cases, each with 10 properties. The clustering technique determines if the observed energy is gamma or not.
- KDD: Composed of regular connections and simulated attacks in a military network environment. With approximately 5 million links, clustering is employed to identify different types of intrusions across all connections. The KDD dataset comprises 33 characteristics for each example.
- Cover type: Indicates the type of cover in US forest cells with a size of 30 by 30 meters. The dataset includes 58,012 entries, each with 54 properties. It predicts the type of tree among seven distinct types in the dataset.

### 3.3. Performance measures

The proposed approach is subject to a thorough evaluation, considering scalability, performance, and clustering quality. Scalability assessment examines its adaptability to various data volumes, ensuring robust functionality across different datasets. Performance metrics delve into computational efficiency and speed, providing insights into the algorithm's responsiveness. Clustering evaluation emphasizes the accuracy in grouping similar data points, contributing to a comprehensive understanding of the proposed approach's effectiveness.

#### 3.3.1. Scalability assessment

To evaluate scalability, measures of speed UP, scale UP, and size UP are employed, defined as follows:

a) Speed up measurement

Speed UP is determined by setting the dataset's size and increasing the CPU core count. Clustering is first executed with a single calculation core and then with m CPU cores, resulting in execution times T1 and Tm, respectively [47]. The following presents the speed UP measurements:

$$SpeedUP = T_1/T_m \tag{7}$$

Where T1 represents the execution time with a single core, and Tm is the execution time with m cores. The optimal algorithm demonstrates speed close to linear scalability.

b) Scale UP measurement

The scale UP measure involves an increase in both the dataset size and the number of CPU cores. Initially, clustering is performed with a single core on a dataset of size N, followed by execution with m cores on a dataset of size m*N [47]. The scale UP value is calculated using the (8):

$$\mathrm{Scale\ UP} = \mathrm{T_{1N}}/\mathrm{T_{mN}} \tag{8}$$

Where $\mathrm{T_{1N}}$: represents the execution time of a dataset of size N on a single core, and $\mathrm{T_{mN}}$ is the execution time of a dataset of size $\mathrm{N} * \mathrm{m}$ on $m$ cores.

When $\mathrm{T_{1N}}$ is close to $\mathrm{T_{mN}}$ the method is considered scalable.

c) Size up measurement:

The size UP metric maintains the number of CPU cores constant while increasing the size of the dataset m times. This metric signifies the algorithm's capability to handle datasets that are m times larger [47]. The size UP measure is calculated using the (9):

$$SizeUP = T_N/T_{mN} \tag{9}$$

$\mathrm{T_N}$ the execution time of dataset of size N.

$\mathrm{T_{mN}}$: the execution time of dataset of size $\mathrm{m} * \mathrm{N}$.

However, the quality of the clustering results is evaluated using the average distance between data objects and their centroids, as defined by (3).

### 3.4. Comparisons and results

Artificial datasets play a pivotal role in assessing the scalability and robustness of the proposed approach, while real datasets are instrumental in evaluating clustering performance. The effectiveness of the approach is determined through a comparative analysis, which involves assessing S-CIPSO against standard PSO (S-PSO) and S-CIPSO excluding the gravity center step (S-CPSO). This comparison serves as a valuable tool for understanding the performance of the proposed approach in relation to various benchmarks, providing a comprehensive perspective on its efficacy.

#### 3.4.1. Related parameters

Conducting tests involves meticulous attention to specific parameters. The swarm, comprising 10 particles, operates under a fixed total of 10 iterations, ensuring consistency in the experimental setup. Throughout the testing process, the inertia weight (w) remains constant at 0.72, providing stability to the algorithm's performance. Additionally, the acceleration coefficients (c1, c2) are steadfastly set at 1.49, contributing to the controlled environment of the experiments. Across all datasets, a uniform total number of clusters (k) is systematically configured at 5, with exceptions noted for the "magic" dataset, where it is constrained to 2, and the "cover type" dataset, where a specific configuration of 7 clusters is implemented.

### 3.4.2. Clustering quality results

Table 3 presents the results of the clustering quality obtained by our approaches in comparison to other algorithms. The findings indicate that our algorithms outperform others in terms of clustering quality. This improvement is attributed to the enhancements introduced into the standard PSO, enabling S-CPSO and S-CIPSO to effectively explore the search space. Notably, S-PSO maintains a comparable quality to PSO. Additionally, it is observed from the table that S-CIPSO significantly enhances the clustering quality compared to S-CPSO. Hence, the implementation of the step that calculates the center of gravity of the best global position emerges as a crucial choice to enhance the quality of clustering.

Table 3. Comparison of the fitness value

| Datasets | PSO | S-PSO | S-CPSO | S-CIPSO |
|---|---|---|---|---|
| Magic | 67.016035 | 82.160078 | 40.938492 | 40.930279 |
| KDD | 252558.92351 | 19412.91393 | 20023.16916 | 14066.84059 |
| Cover type | 989.718507 | 1169.723182 | 464.810866 | 464.370944 |

### 3.4.3. Performance results

Table 4 presents the performance results of the proposed approaches in comparison to other algorithms. According to this table, all algorithms implemented under the Spark environment (S-PSO, S-CPSO, and S-CIPSO) demonstrate superior performance compared to the standard PSO on all datasets, except for the “magic” dataset. This discrepancy is explained by considering that “magic” is relatively small compared to the other datasets, rendering parallel processing less advantageous. In such cases, parallel processing becomes more resource-intensive and time-consuming. It is evident from the results that S-CPSO outperforms S-PSO across all datasets. This improvement is attributed to the utilization of the logistic map, enhancing the search mechanism without adding complexity to the algorithm. Furthermore, the table illustrates that the difference in execution time between S-PSO and S-CIPSO diminishes as the dataset size increases.

Table 4. Running time in seconds (s)

| Datasets | PSO | S-PSO | S-CPSO | S-CIPSO |
|---|---|---|---|---|
| Magic | 3.939328 | 35.031629 | 28.455660 | 41.740645 |
| KDD | 2346.306556 | 1307.707522 | 1005.495059 | 1328.462582 |
| Cover type | 477.549305 | 321.658103 | 286.035391 | 362.224696 |

### 3.4.4. Evaluation of the S-CIPSO scalability and robustness

The running time of the suggested approach on all the used artificial datasets is shown in Figure 2. From Figure 2, it is clear that increasing the number of cores reduces the execution time for all datasets. For example, the running time of D5M with 1 core is 4,935 seconds, and with 5 cores, it is 1,104 seconds, which means it has decreased more than 4 times.

To assess the speed UP of the proposed method, the dataset size is held constant while the number of cores is increased from 1 to 5. Figure 3 illustrates the speed results of the tested datasets. The outcomes demonstrate that the suggested approach has good speed UP. In fact, for all datasets, when the number of cores varies from 1 to 3, the speed UP results are nearly linear, making them a very strong speed UP. However, the performance of the suggested approach starts to deteriorate, and the speed UP deviates from linearity when the number of cores is higher than 4. This is interpreted by the longer communication times needed to handle the increased core count.

To assess the scale UP of the presented method, both the dataset size and the number of cores is increased. The examination of scale UP utilizes all tested artificial datasets with cores set to 1, 2, 3, 4, and 5, respectively. The results are depicted in Figure 4. As previously mentioned, the ideal algorithm exhibits Scale UP values extremely close to or equal to 1. In our case, the suggested method demonstrates a very good Scale UP, indicating that S-CIPSO adapts well to scale UP values between 1 and 0.90.

Figure 5 illustrates the results obtained by S-CIPSO for 1, 2, 3, 4, and 5 cores, respectively. The outcomes depict a favorable size UP for the proposed method. For instance, on a single core, the size UP results exhibit nearly linear improvement.

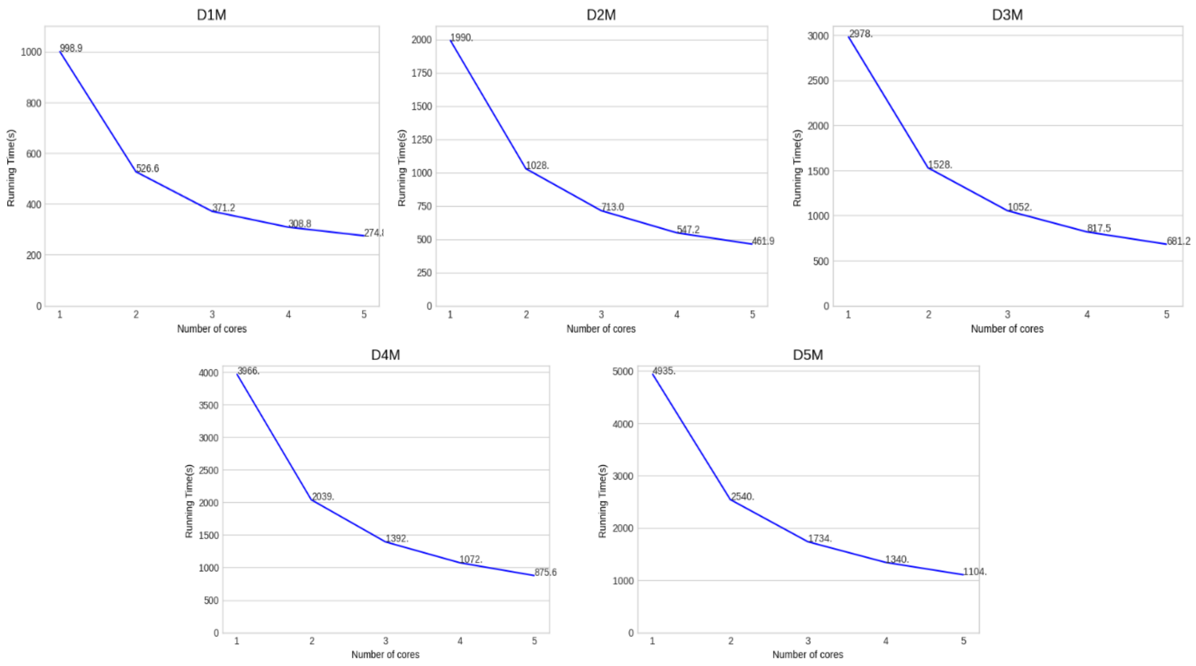


Figure 2. Execution time of S-CIPSO on the different artificial datasets

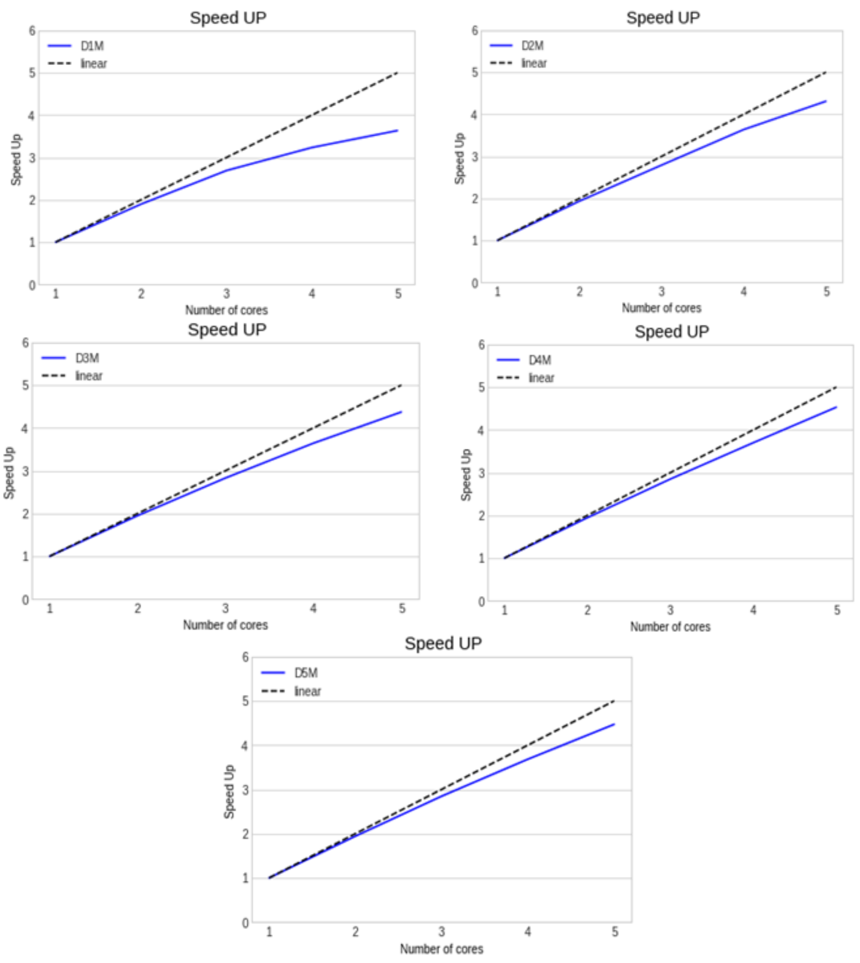


Figure 3. Speed up results of S-CIPSO on the different artificial datasets

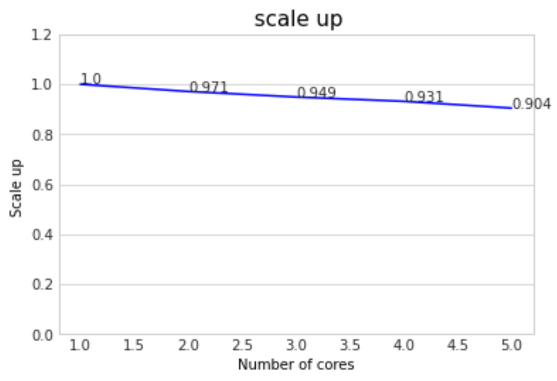


Figure 4. Scale up results of S-CIPSO

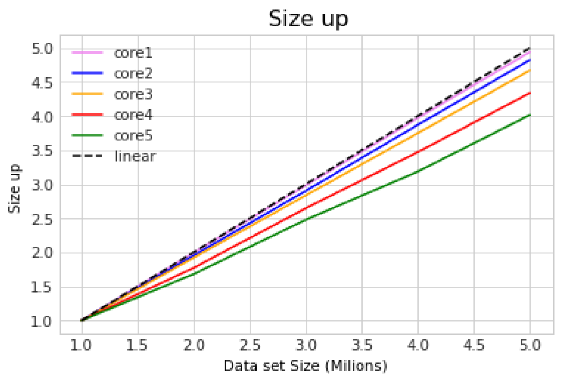


Figure 5. Size up results of S-CIPSO

## 4. CONCLUSION

This study introduces a novel algorithm, S-CIPSO, designed for the clustering of big data within the Spark Framework. Initially, the conventional PSO algorithm is enhanced by incorporating a chaotic map (S-CPSO). Subsequently, an additional step is introduced where the gravity center of the best position is calculated, leading to the development of S-CIPSO. The implementation is carried out using the Python language. To evaluate clustering results, two types of datasets real and artificial are employed. Real datasets are utilized to assess clustering quality and performance, while artificial datasets are reserved for evaluating the scalability and robustness of S-CIPSO. The test results reveal that S-CPSO outperforms all other algorithms, demonstrating superior clustering quality compared to both S-PSO and the standard PSO. This improvement is attributed to the stochastic searches of the logistic map, effectively exploring the search area with features like non-repetition and ergodicity without adding complexity to the algorithm. Furthermore, S-CIPSO significantly enhances the clustering quality of S-CPSO, emphasizing the importance of determining the gravity center of the best global position to increase clustering quality. Experimental results affirm the scalability of S-CIPSO, showcasing its suitability for big data clustering. In our future endeavors, we plan to extend the application of the proposed approach to image segmentation.

## REFERENCES

[1] W. Inoubli, S. Aridhi, H. Mezni, M. Maddouri, and E. M. Nguifo, “An experimental survey on big data frameworks,” *Future Generations Computer Systems: FGCS,* vol. 86, pp. 546–564, 2018, doi: 10.1016/j.future.2018.04.032.
[2] M. Chen, S. Mao, and Y. Liu, “Big data: a survey,” *Mobile Networks and Applications*, vol. 19, no. 2, pp. 171–209, 2014, doi: 10.1007/s11036-013-0489-0.
[3] M. H. Iqbal and T. R. Soomro, “Big data analysis: Apache Storm perspective,” *International Journal of Computer Trends and Technology*, vol. 19, pp. 9–14, 2015, doi: 10.14445/22312803/IJCTT-V19P103.
[4] N. Khan *et al.*, “Big data: survey, technologies, opportunities, and challenges,” *The Scientific World Journal*, vol. 2014, pp. 1–18, 2014, doi: 10.1155/2014/712826.
[5] J. Nereu, A. Almeida, and J. Bernardino, “Big data analytics: a preliminary study of open source platforms,” *Proceedings of the 12th International Conference on Software Technologies,* vol. 1, pp. 435–444, 2017, doi: 10.5220/0006470104350440.
[6] A. Oussous, F. Z. Benjelloun, A. Ait-Lahcen, and S. Belfkih, “Big data technologies: a survey,” *Journal of King Saud University-Computer and Information Sciences*, vol. 30, no. 4, pp. 431–448, 2018, doi: 10.1016/j.jksuci.2017.06.001.
[7] S. Sagiroglu and D. Sinanc, “Big data: A review,” *in IEEE 2013 International Conference on Collaboration Technologies and Systems (CTS),* 2013, doi: 10.1109/CTS.2013.6567202.
[8] M. A. Mahdi and K. M. Hosny, “Scalable clustering algorithms for big data: a review,” *IEEE Access: Practical Innovations, Open Solutions*, vol. 9, pp. 80 015–80 027, 2021, doi: 10.1109/ACCESS.2021.3084057.
[9] S. A. S. Shirkhorshidi, T. T. Y. Wah, and B. Herawan, “Big data clustering: a review,” *ICCSA,* 2014, doi: 10.1007/978-3-319-09156-3_49.
[10] S. I. Boushaki, N. Kamel, and O. Bendjeghaba, “A new quantum chaotic cuckoo search algorithm for data clustering,” *Expert Systems With Applications*, vol. 96, pp. 358–372, Apr. 2018, doi: 10.1016/j.eswa.2017.12.001.
[11] S. I. Boushaki, N. Kamel, and O. Bendjeghaba, “High-dimensional text datasets clustering algorithm based on cuckoo search and latent semantic indexing,” *Journal of Information and Knowledge Management*, vol. 17, 2018, doi: 10.1142/s0219649218500338.
[12] S. I. Boushaki, N. Kamel, O. Bendjeghaba, “Biomedical document clustering based on accelerated symbiotic organisms search algorithm,” *International Journal of Swarm Intelligence Research*, vol. 12, no. 4, pp. 169-185, 2021, doi: 10.4018/IJSIR.2021100109.
[13] A. Fahad *et al.*, “A survey of clustering algorithms for big data: Taxonomy and empirical analysis,” *IEEE Transactions on Emerging Topics in Computing*, vol. 2, no. 3, pp. 267–279, 2014, doi: 10.1109/TETC.2014.2330519.
[14] M. Hai, S. Zhang, L. Zhu, and Y. Wang, “A survey of distributed clustering algorithms,” *International Conference on Industrial Control and Electronics Engineering,* 2012, doi: 10.1109/ICICEE.2012.303.
[15] T. Hanghang and U. Kang, “Big data clustering,” *Data Clustering,* pp. 259–276, 2018, doi: 10.1201/9781315373515-11.

[16] M. Khalid and M. M. Yousaf, "A comparative analysis of big data frameworks: an adoption perspective," *Applied Sciences*, vol. 11, no. 22, 2021, doi: 10.3390/app112211033.

[17] A. Siddiqa, A. Karim, and A. Gani, "Big data storage technologies: a survey," *Frontiers of Information Technology and Electronic Engineering*, vol. 18, pp. 1040–1070, 2017, doi: 10.1631/FITEE.1500441.

[18] D. Singh and C. Reddy, "A survey on platforms for big data analytics," *Journal of Big Data*, vol. 2, no. 1, pp. 1–20, 2015, doi: 10.1186/s40537-014-0008-6.

[19] C. Tsai, C. F. Lai, H. C. Chao, and A. V. Vasilakos, "Big data analytics: a survey," *Journal of Big Data*, no. 1, pp. 1–32, 2015, doi: 10.1186/s40537-015-0030-3.

[20] J. Wang, Y. Yang, T. Wang, R. Sherratt, and J. Zhang, "Big data service architecture: a survey," *Journal of Internet Technology*, vol. 21, no. 2, pp. 393–405, 2020, doi: 10.3966/160792642020032102000.

[21] X. Zhang and F. Xu, "Survey of research on big data storage," *12th International Symposium on Distributed Computing and Applications to Business*, 2013, doi: 10.1109/DCABES.2013.21.

[22] N. Ahmed, A. L. C. Barczak, T. Susnjak, and M. Rashid, "A comprehensive performance analysis of apache hadoop and apache spark for large scale data sets using HiBench," *Journal of Big Data*, vol. 7, no. 1, pp. 1–20, 2020, doi: 10.1186/s40537-020-00388-5.

[23] D. K. Sharmila, D. S. Kamalakkannan, R. Devi, and D. C. Shanthi, "Big data analysis using apache hadoop and spark," *International Journal of Recent Technology and Engineering*, vol. 8, no. 2, pp. 167–170, 2019, doi: 10.35940/ijrte.A2128.078219.

[24] S. Tang, B. He, C. Yu, Y. Li, and K. Li, "A survey on spark ecosystem: Big data processing infrastructure, machine learning, and applications," *IEEE Transactions on Knowledge and Data Engineering*, pp. 1–21, 2020, doi: 10.1109/TKDE.2020.2975652.

[25] X. Cui, P. Zhu, X. Yang, C. Li, and Ji, "Optimized big data K-means clustering using MapReduce," *The Journal of Supercomputing*, vol. 70, pp. 1249–1259, 2014, doi: 10.1007/s11227-014-1225-7.

[26] R. Alguliyev, R. M. Alıguliyev, and L. Sukhostat, "Efficient algorithm for big data clustering on single machine," *CAAI Transactions on Intelligence Technology*, vol. 5, no. 1, pp. 9–14, Jan. 2020, doi: 10.1049/trit.2019.0048.

[27] P. P. Prajapati, A. J. Kshatriya, D. N. Patel, S. K. Gonsai, H. B. Tank, and K. R. Sheth, "Comparative analysis of meta heuristics algorithm for differential amplifier design," *Bulletin of Electrical Engineering and Informatics (BEEI)*, vol. 12, no. 6, pp. 3395–3401, 2023, doi: 10.1007/978-3-031-29857-8_71.

[28] H. S. Maharana and S. K. Dash, "Quantum behaved artificial bee colony based conventional controller for optimum dispatch," *International Journal of Electrical and Computer Engineering*, vol. 13, no. 2, p. 1260, 2023, doi: 10.11591/ijece.v13i2.pp1260-1271.

[29] O. M. Neda, "A new hybrid algorithm for solving distribution network reconfiguration under different load conditions," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 20, no. 3, p. 1118, 2020, doi: 10.11591/ijeecs.v20.i3.pp1118-1127.

[30] Z. Benmounah, S. Meshoul, and M. Batouche, "Scalable differential evolutionary clustering algorithm for big data using map-reduce paradigm," *International Journal of Applied Metaheuristic Computing*, vol. 8, no. 1, pp. 45–60, 2017, doi: 10.4018/IJAMC.2017010103.

[31] S. E. Hashemi, M. Tavana, and M. Bakhshi, "A new particle swarm optimization algorithm for optimizing big data clustering," *SN Computer Science*, vol. 3, no. 4, 2022, doi: 10.1007/s42979-022-01208-8.

[32] M. Sherar and F. Zulkernine, "Particle swarm optimization for large- scale clustering on apache spark," in *IEEE Symposium Series on Computational Intelligence (SSCI)*, 2017, pp. 1–8, doi: 10.1109/SSCI.2017.8285208.

[33] A. Sinha and P. K. Jana, "A hybrid MapReduce-based k-means clustering using genetic algorithm for distributed datasets," *The Journal of Supercomputing*, vol. 74, 2018, doi: 10.1007/s11227-017-2182-8.

[34] R. M. Alguliyev, R. M. Aliguliyev, and F. J. Abdullayeva, "PSO+K- means algorithm for anomaly detection in big data," *Statistics Opti- mization and Information Computing*, vol. 7, no. 2, pp. 348–359, 2019, doi: 10.19139/soic.v7i2.623.

[35] M. Moslah, M. A. B. Hajkacem, and N. Essoussi, "Spark-based design of clustering using particle swarm optimization," *Clustering Methods for Big Data Analytics*, pp. 91–113, 2019, doi: 10.1007/978-3-319-97864-2_5.

[36] V. Ravuri and S. Vasundra, "Moth-flame optimization-bat optimization: Map-reduce framework for big data clustering using the moth-flame bat optimization and sparse fuzzy C-means," *Big Data*, vol. 8, no. 3, pp. 203–217, 2020, doi: 10.1089/big.2019.0125.

[37] M. Q. Bashabsheh, L. Abualigah, and M. Alshinwan, "Big data analysis using hybrid meta-heuristic optimization algorithm and MapReduce framework," *in Studies in Computational Intelligence, Cham: Springer International Publishing*, 2022, pp. 181–223, doi: 10.1007/978-3-030-99079-4_8.

[38] N. Q. Ann, D. Pebrianti, M. F. Abas, and L. Bayuaji, "Automated-tuned hyper-parameter deep neural network by using arithmetic optimization algorithm for Lorenz chaotic system," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 13, no. 2, p. 2167, 2023, doi: 10.11591/ijece.v13i2.pp2167-2176.

[39] S. H. Shri and A. F. Mijbas, "Chaotic theory incorporated with PSO algorithm for solving optimal reactive power dispatch problem of power system," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 22, no. 3, p. 1739, 2021, doi: 10.11591/ijeecs.v22.i3.pp1739-1747.

[40] W. A. Wali, "Application of particle swarm optimization with ANFIS model for double scroll chaotic system," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 11, no. 1, p. 328, 2021, doi: 10.11591/ijece.v11i1.pp328-335.

[41] Z. M. Gao, J. Zhao, and Y. J. Zhang, "Review of chaotic mapping enabled nature-inspired algorithms," *Mathematical Biosciences and En- gineering: MBE*, vol. 19, no. 8, pp. 8215–8258, 2022, doi: 10.3934/mbe.2022383.

[42] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization: An overview," *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007, doi: 10.1007/s11721-007-0002-0.

[43] M. M. Saeed, M. M, Z. Aghbariand, and M. Alsharidah, "Big data clustering techniques based on spark: a literature review," *PeerJ Computer Science*, vol. 6, no. e321, pp. 2020–2020, doi: https://doi.org/10.7717/peerj-cs.321.

[44] G, D. Fatta, and S. A. Ghamdi, "Efficient clustering techniques on Hadoop and spark," *International Journal of Big Data Intelligence*, vol. 6, no. 1, 2019, doi: 10.1504/IJBDI.2019.100898.

[45] N. Belbachir, M. Zellagui, A. Lasmari, C. Z. El-Bayeh, and B. Bekkouche, "Optimal integration of photovoltaic distributed generation in electrical distribution network using hybrid modified PSO algorithms," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 24, no. 1, p. 50, 2021, doi: 10.11591/ijeecs.v24.i1.pp50-60.

[46] M. Z. M. Tumari, A. F. Z.Abidin, A. S. R. A. Subki, A. W. A. Aziz, M. S. Saealal, and M. A. Ahmad, "Liquid slosh control by implementing model-free PID controller with derivative filter based on PSO," *Indonesian Journal of Electrical Engineering and Computer Science (IJEECS)*, vol. 18, no. 2, p. 750, 2020, doi: 10.11591/ijeecs.v18.i2.pp750-758.

[47] X. Xu, J. Jäger, and H. P. Kriegel, "A fast-parallel clustering algorithm for large spatial databases," *Data Mining Discovery*, vol. 3, pp. 263–290, 1999, doi: 10.1023/A:1009884809343.

## BIOGRAPHIES OF AUTHORS

**Saida Ishak Boushaki** is an Associate Professor in the Department of Computer Science at the University M'hammed Bouguera of Boumerdes, Algeria. She earned her Magister and engineering degrees in Computer Science from the same university in 2003 and 2007, respectively. Since 2011, she has been a member of the Laboratory of Research in Artificial Intelligence (LRIA) at the University of Science and Technology Houari Boumediene (USTHB). She undertook an 18-month internship at ETS (École de technologie supérieure), Montreal, Canada, starting in October 2014. In 2018, she obtained her Ph.D. degree in Computer Science from USTHB. Her primary interests include data mining, big data clustering, information retrieval, formal methods modeling, and the application of evolutionary computation to solve optimization problems. She can be contacted via email at: s.boushaki@univ-boumerdes.dz.

**Brahim Hadj Mahammed** is a Software Engineer with expertise in web development, datamining, and big data. With 3 years of experience as a freelance web developer, he has delivered numerous successful projects. Brahim holds a Master's degree in Software Engineering and Data Processing, equipping him with a solid educational background. He excels in programming languages such as HTML, CSS, JavaScript, and Python, and has a deep understanding of datamining techniques and big data technologies. His ability to extract valuable insights from complex datasets makes him a valuable asset in leveraging data for informed decision-making. Combining his web development skills with datamining expertise, Brahim delivers innovative solutions that maximize the potential of data-driven applications. He can be contacted at email: brahimhm98@gmail.com.

**Omar Bendjeghaba** was born in Algeria. He Hold Magister Degree (Master of Engineering with dissertation) from The FHC Formally INHC. He received his Ph.D. degree from the University of M'hamed Bougarra Boumerdes in 2010. He is currently member at the research laboratory (LREEI). His research activities include control, planning and operations of power systems, as well as evolutionary computation theory and applications in power systems. He can be contacted at email: bendjeghaba@univ-boumerdes.

**Messaoud Mosbah** is a Software Engineer with a Master's degree in the field. He has been working as a software developer for some time and has a passion for machine learning and big data. He is constantly striving to improve his skills and stay up-to-date with the latest advancements in this field. He can be contacted at email: m.mesd16@gmail.com.